Abstract

Dit artikel introduceert een architecturaal model voor de ontwikkeling van en interactie met web services. Hierbij wordt een web service beschreven als bestaande uit bedrijfsobjecten die participeren in business events.
Het event-concept wordt vervolgens gebruikt als coördinatiemechanisme, dat de compositie en choreografie overheen meerdere web services ondersteunt.

Een event-gebaseerde arch

Wilfried Lemahieu

Web services als technologie voor EAI en B2Bi

Eén van de belangrijkste ICT-tendensen van de laatste jaren is de nood aan *integratie*. Integratie van de informatiesystemen binnen éénzelfde onderneming wordt omschreven als EAI (Enterprise Application Integration). Recent maakte echter ook het concept van de *extended enterprise* opgang, waarbij verschillende onafhankelijke ondernemingen (bijvoorbeeld producenten en distributeurs) hun bedrijfsprocessen integreren tot één geoptimaliseerde waardeketting. Dit noodzaakt tot op zekere hoogte ook de koppeling van hun informatiesystemen. Wanneer de informatiesystemen van verschillende onafhankelijke bedrijven worden geïntegreerd, spreekt men van B2Bi (Business-to-Business interaction). Bij EAI en zeker bij B2Bi wordt de koppeling tussen de onderscheiden systemen best zo "los" mogelijk gehouden; de onderlinge afhankelijkheid blijft daardoor minimaal, wat een maximale flexibiliteit garandeert. Web services lijken hierbij uit te groeien tot dé technologie voor de losse koppeling van systemen met behulp van het Internet.

Web services kunnen omschreven worden als zelfstandige software-componenten, die een welbepaalde (business-) functionaliteit aanbieden, die door andere applicaties of services kan worden aangesproken over het Internet. Interactie met een web service gebeurt met behulp van SOAP (Simple Object Access Protocol). Elke SOAP-aanroep specificeert een geparameteriseerde *operatie* die door een bepaalde web service moet uitgevoerd worden. SOAP heeft als grote voordeel dat het een zeer “licht” protocol is, dat kan gebruikt worden overheen het gehele Internet. "Zwaardere" alternatieven zoals CORBA, RMI en DCOM leveren ondermeer firewall-problemen op en beperken zich daardoor noodgedwongen tot het lokale netwerk van eenzelfde onderneming. Web services en SOAP vormen op deze wijze de de- facto standaard voor B2Bi, hoewel ze ook opgang maken voor EAI.

Waar het web services concept dus veelbelovend is, beperken huidige implementaties zeker op gebied van B2Bi zich doorgaans tot vrij eenvoudige, niet-transactionele systemen. Voorbeelden zijn muntconversie-toepassingen, on-line beursberichten enzovoort. Het is momenteel echter vrijwel onmogelijk om web services complexe bedrijfstransacties te laten uitvoeren, waarin verscheidene partners participeren. Een eerste reden is dat web services, in plaats van zich te baseren op een degelijke *architectuur*, nog al te vaak "ad hoc" worden ontwikkeld, wat nauwelijks toelaat om complexe toepassingen te realiseren. Een tweede reden is dat het onderliggende SOAP protocol in essentie een één-op-één communicatiemechanisme is, dus in sé minder geschikt is om de simultane interactie tussen meer dan twee services te coördineren. Sectie 2 van dit artikel komt hieraan tegemoet door een architecture-driven benadering van web service ontwikkeling voor te stellen, waarbij de interactie verloopt door de gezamenlijke participatie van *bedrijfsobjecten in business events*. Een robuuste architectuur moet het mogelijk maken om ook de ontwikkeling van meer complexe, transactionele web service toepassingen tot een goed einde te brengen. Sectie 3 beschrijft hoe events als coördinerend mechanisme kunnen optreden, wat ook de *compositie* en *choreografie* van web services eenvoudiger maakt.

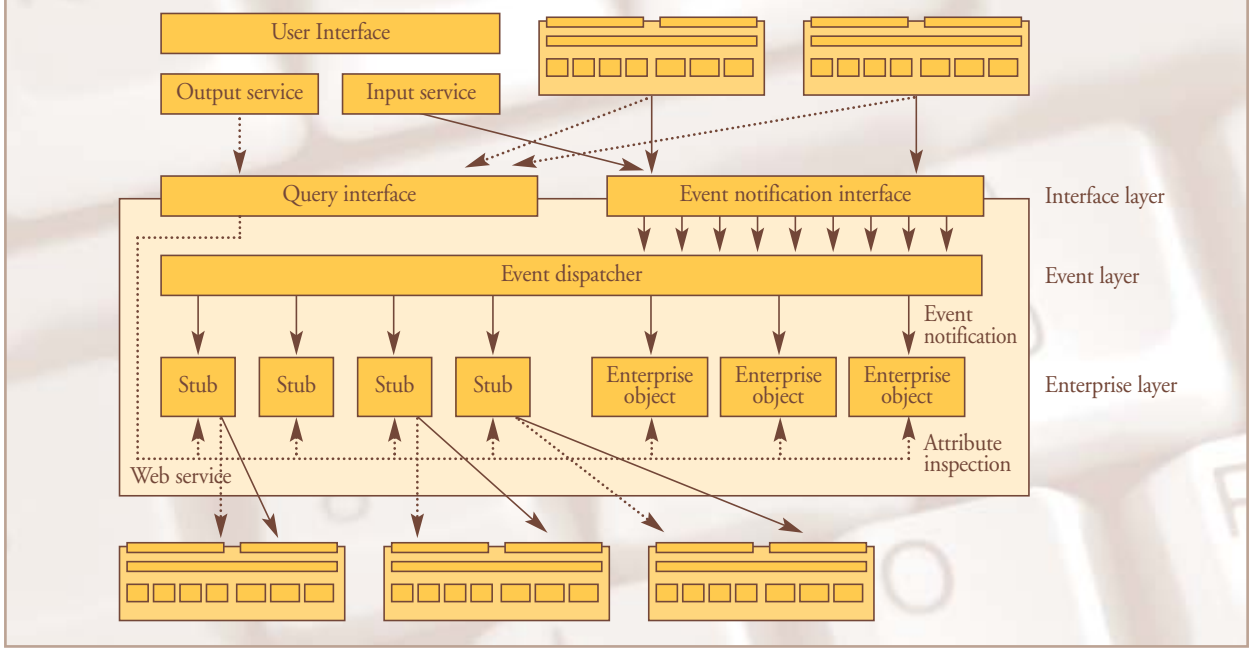
Een architecture-driven benadering van web service ontwikkeling

Wij stellen een traject voor web service ontwikkeling voor in drie fasen. Een eerste fase behelst, zoals elk kwaliteitsvol ICT-project, een analyse- en designfase. Deze is gebaseerd op de MERODE-methodologie [1]. Het resultaat is een *bedrijfsmodel*, dat enerzijds bedrijfsobjecten (enterprise objects) bevat zoals KLANT, PRODUCT, LEVERANCIER, … en anderzijds *bedrijfsgebeurtenissen* (business events) die op deze objecten kunnen inwerken, bijvoorbeeld AANKOOP, LEVERING, BETALING, … Het model beschrijft eveneens de voorwaarden (precondities) die door een bepaald object type aan een bepaald event type kunnen opgelegd worden (bijvoorbeeld: bij een AANKOOP-event vereist het betrokken PRODUCT-object dat *ordergrootte* ≤ *voorraad)* en de gevolgen *(postcondities)* die een bepaald event type heeft op een bepaald object type (bijvoorbeeld: *voorraad* → *voorraad* – *ordergrootte*). Deze precondities en postcondities worden afgeleid uit de *bedrijfsregels* (business rules) die impliciet of expliciet met de bedrijfsprocessen zijn geassocieerd.

In een tweede stap wordt dit bedrijfsmodel verrijkt tot een *architectuurmodel*. Een architectuurmodel is nog steeds onafhankelijk van specifieke software-technologieën en -platformen, maar houdt wel rekening met de *soort* implementatie-omgeving: sterk of los gekoppeld, gelokaliseerd of gedistribueerd enz. Een voorbeeld van een gecentraliseerde, sterk gekoppelde architectuur wordt besproken in [2]. Een architectuurmodel voor een los gekoppelde, gedistribueerde omgeving zoals het geval is bij web services wordt getoond in figuur 1.

Een volledige beschrijving van deze architectuur is te vinden in [3]. Daarin wordt ook de derde stap in de ontwikkelingsfase besproken, waarbij het architectuurmodel wordt vertaald naar een *implementatiemodel* op basis van welbepaalde software-technologieën. In het geval van web services is dit met behulp van SOAP en WSDL (Web Service Description Language). In wat volgt bespreken wij kort de belangrijkste elementen van het architectuurmodel.

Fig. 1. Architectuurmodel voor een web services omgeving



Belangrijk is dat, in tegenstelling tot “traditionele” web services, een onderscheid wordt gemaakt tussen twee soorten interfaces tot de web service; de *query interface* dient om attribuutwaarden van de bedrijfsobjecten te lezen, m.a.w. om informatie op te vragen aan de web service. Bijvoorbeeld: welke producten beschikbaar zijn in een on-line winkel. Via een query interface worden nooit gegevens gewijzigd, daarom is dergelijke aanroep nooit het voorwerp van een transactie.

De event *notification interface* dient om de web service op de hoogte te brengen van een bedrijfsgebeurtenis, bij-

voorbeeld: een AANKOOP. De “event notification” wordt doorgegeven aan de event dispatcher, die het event “uitzendt” naar alle betrokken bedrijfsobjecten in de web service. Elk bedrijfsobject kan individueel precondities opleggen aan het event; een KLANT-object zal bijvoorbeeld nagaan of de betrokken klant geen wanbetaler is, een PRODUCT-object zal nagaan of het product wel voldoende in voorraad is enzovoort. Indien aan alle precondities is voldaan, reageren de bedrijfsobjecten op het event door hun attribuutwaarden aan te passen. Zo zal een KLANT-object zijn lijst met uitstaande orders aanpassen, het PRODUCT-object zal zijn stock wijzigen enzovoort. Aangezien een aanroep naar de event notification interface dus wijzigingen in de status van de objecten tot gevolg heeft, is deze wél het voorwerp van een transactiecontext. De event dispatcher coördineert de transactie en zorgt dat enkel als in alle objecten aan alle precondities voldaan is, de transactie effectief mag plaatsvinden. Merk op dat in de figuur een aanroep naar de query-interface niet via de event dispatcher passeert: bij een query is zoals gezegd geen transactie-context betrokken.

in figuur 2. Een bepaalde web service wordt op de hoogte gebracht van een bedrijfsgebeurtenis via een aanroep naar zijn event notification interface (1). Dit event wordt “gedispatcht” naar alle lokale bedrijfsobjecten en stub objecten (2). Elk stub object propageert het event naar de door hem vertegenwoordigde web service (3). In elke remote web service worden de eigen bedrijfsobjecten op de hoogte gebracht via diens event dispatcher (4). Ook hier kunnen eventueel weer stub objecten voorkomen die het event verder propageren naar andere web services. Alle betrokken bedrijfsobjecten in de verschillende web services kunnen precondities opleggen en zullen hun status wijzigen als aan alle precondities is voldaan (5). De consistentie van de transactie wordt gewaarborgd door de samenwerking van de respectievelijke event dispatchers en stub objecten.

Op deze wijze wordt het mogelijk om een web service een complexe taak te laten vervullen, door via de stub-objecten deelopdrachten (eventueel recursief) te delegeren naar externe services. Een typisch voorbeeld is een “reisbureau” aangezien enkel de event notification interface op transactionele aanroepen gericht is. Daarnaast leidt het onderscheid tussen operaties die “lezen” van de web service (via de query interface) en operaties die “schrijven” naar de web service (via de event notification interface) tot beter onderhoudbare en herbruikbare software, zoals geponeerd in het Command-Query Separation Principle. Dit alles is in detail beschreven in [4].

Tevens wordt het veel eenvoudiger om de samenwerking van verscheidene web services in complexe business transacties te coördineren. De simultane propagatie van events tussen de deelnemende services vormt hierbij een n-op-n coördinatiemechanisme overheen de standaard één-op-één SOAP interactie. Het choreografie-aspect is gedistribueerd overheen de deelnemende bedrijfsobjecten in de vorm van precondities en postcondities, zodat het gedrag van de services de bedrijfsprocessen van de participerende ondernemingen weerspiegelt.

itectuur voor web services

Web service compositie en choreografie

De voorgestelde architectuur laat niet enkel toe om een individuele, transactionele, web service op systematische wijze te ontwerpen, ze vereenvoudigt tevens de *compositie* en *choreografie* overheen meerdere web services. Hier spelen de *stub objecten* een belangrijke rol. Een stub object kan beschouwd worden als een “vertegenwoordiger” van een andere service. Een stub object in de ene service zal een relevante bedrijfsgebeurtenis *propageren* naar de event notification interface van de remote service die hij vertegenwoordigt. Het volledige event-gebaseerde interactiemechanisme verloopt in vier fasen, zoals voorgesteld

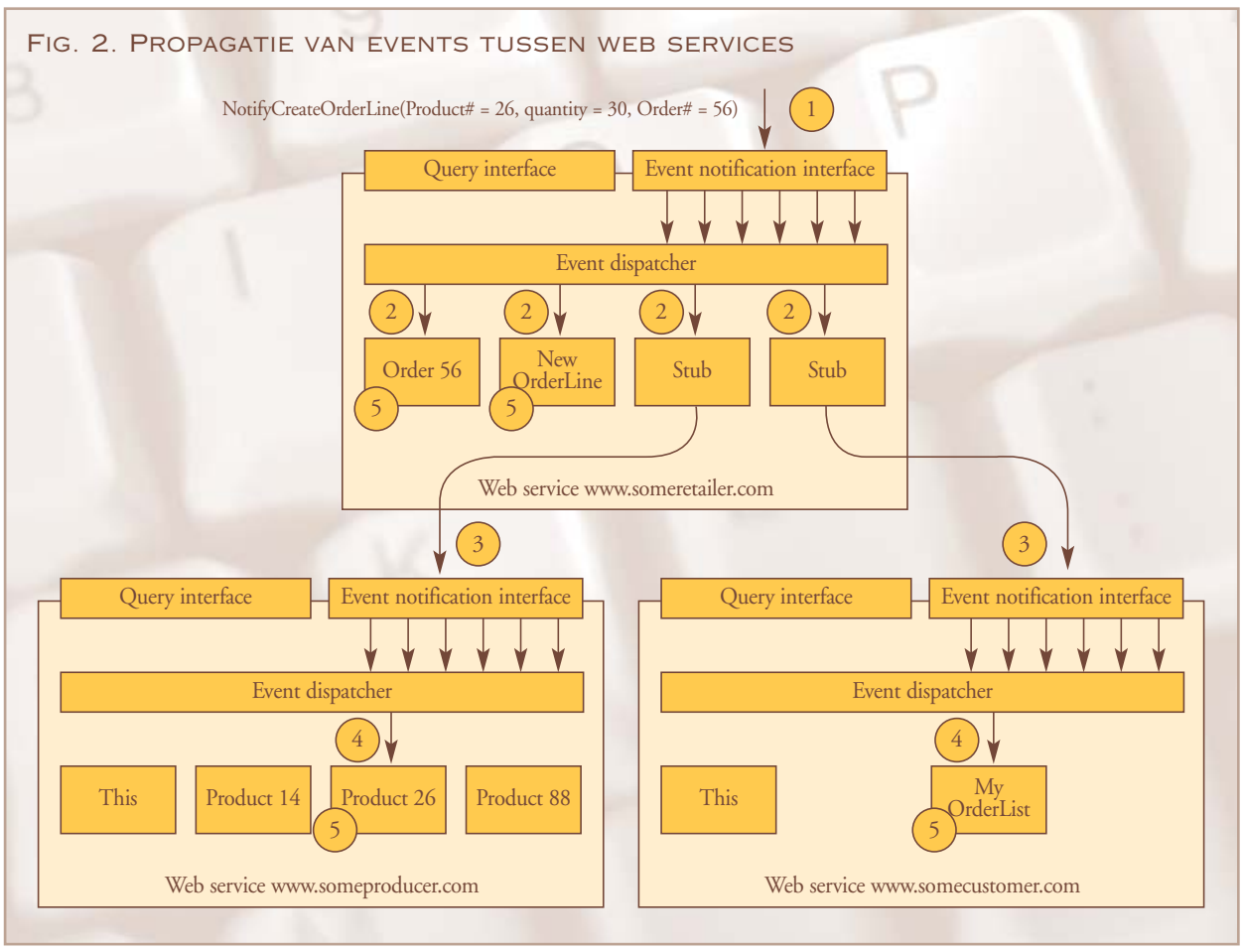
web service, die de gecombineerde boeking van een vlucht, hotel en huurauto uitbesteedt aan andere web services, waarbij het geheel als één transactie wordt opgevat.

Besluiten

De in dit artikel voorgestelde architectuur maakt het mogelijk om web service-ontwikkeling op een systematische wijze te benaderen. Hierbij wordt in de uitwendige specificaties van de web service (de interfaces) een duidelijk onderscheid gemaakt tussen wat de service doet (reageren op business events) en welke *informatie* de service aanbiedt (via de query interface). Dit maakt de specificatie van transactionele systemen veel eenvoudiger,

Wilfried Lemahieu
is docent aan het Departement Toegepaste Economische Wetenschappen van de K.U.Leuven, vakgroep Beleidsinformatica.
E-mail: wilfried.lemahieu@econ.kuleuven.ac.be

Fig. 2. Propagatie van events tussen web services



Referenties:

1. Snoeck M., Dedene G., Verhelst M., Depuydt A. M., Object-oriented Enterprise Modeling with MERODE, Leuven University Press, Leuven (1999)
2. Lemahieu W., Snoeck M., Michiels C., An Enterprise Layer Based Approach to Application Service Integration, Business Process Management Journal, Special Issue on Application Service Provision (verschijnt medio 2003)
3. Lemahieu W., Snoeck M., Michiels C., Goethals F., Dedene G., Vandenbulcke J., A Model Driven, Layered Architecture for Web Service Development, K.U.Leuven – F.E.T.E.W. internal research report, currently under review at IEEE Computer – Special Issue on Web Services Computing
4. Lemahieu W., Snoeck M., Michiels C., Goethals F., An Event Based Approach to Web Service Design and Interaction, Proceedings of the 2003 Asia Pacific Web Conference (APWeb2003), Xi'an, LNCS 2642